Lec : 11 "AI : Constraint Satisfaction Problems"

→ For what we search ?

- we can classify search Problems into 2 types according to what we search For → Planning : when we search For the optimal path to goal.
  → Identification : when we search For the goal itself.

[1] Planning : " Sequences of actions "
- → The path to the goal is the important thing
- → Paths have different costs, depths
- → Heuristics give problem specific توجيه guidance

[2] Identification : " Assignments to Variables "
- → The goal itself is important, not the path.
- → All paths at the same depth (equal cost For all paths)
- → Constraint Satisfaction Problems are Specialized For Identification Problems.

→ Generally The World Contains
1. Single Agent
2. Actions.
3. States.
4. Discrete State Space.

★ Standard (Planning) Search Problems:

- State : is a "black box"
- Goal test : Can be any function over states.
  Function that makes us move from state to another
- Successor function can also be anything

★ Constraint Satisfaction Problems (CSPs) :

- A special subset of search problems (Identificat
- State : is defined by variables X_i with values from a domain D (sometimes D depends on i)
- Goal test : is a set of constraints specifing allowable combinations of values for subsets of variables.

Ex: CSPs of :

1. State: ⇒ variables X, Y, Z
2. Domain: ⇒ {1, 2, 3}
   This mean that X can take values equal to 1 or 2 or 3 and Y, Z can take values equal to 1 or 2 or 3.
3. Goal test (constraints): X = 1 , Y = 2 , Z = 3
   This mean that the goal we search for is the state of X with value equal to 1 and Y = 2 , Z = 3

- CSPs is a simple example of a formal representation language.
  → easy to write a code of CSPs using any programming language. Just identify variables, Domain and put constraints

- CSPs allows useful general-purpose algorithms with more power than Standard Search algorithms.

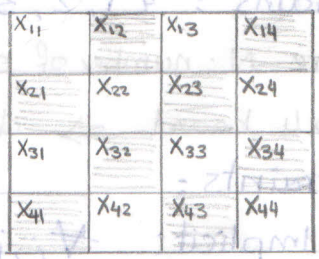
→ we will study CSPs

Example: Map Coloring

- Variables : WA, NT, Q, NSW, V, SA, T (cuntries)
- Domain : $D = \{red, green, blue\}$
  → each cuntry can take one of 3 colors red or green or blue.
- Constraints: adjacent regions must have different colors.
  - Implicit : $WA \neq NT$ (Implicit : provide code to compute)
  - Explicit : $(WA, NT) \in \{(red, green), (red, blue), (green, blue)\}$
    (Explicit : provide a list of the legal tuples)
- Solutions are assignments satisfying all constraints, e.g:
  $\{WA = red, NT = green, Q = red, NSW = green, V = red$
  $SA = blue, T = green\}$

Example : N-Queens.

Formulation 1



N* N_queens

NXN board

- Variables : X_{ij} (one square location)
  (16 variable) : $(X_{11}, X_{12}, X_{13}, X_{14}, \ldots, X_{44})$
- Domains : $\{0, 1\}$
  $X_{ij} = 0$ → There is no queen in this location
  $X_{ij} = 1$ → There is a queen in this location.
- Constraints :
  - Explicit $\forall i, j, K \ (X_{ij}, X_{iK}) \in \{(0,0), (0,1), (1,0)\}$
    make sure that each row has no or one queen
    $\forall i, j, K \ (X_{ij}, X_{Kj}) \in \{(0,0), (0,1), (1,0)\}$
    make sure that each column has no or one queen.

$\forall i,j,K \ (X_{ij}, X_{i+K, j+K}) \in \{(0,0),(0,1),(1,0)\}$

make sure that each قطر has no or one queen

$\forall i,j,K \ (X_{ij}, X_{i+K, j-K}) \in \{(0,0),(0,1),(1,0)\}$

make sure that each قطر عكسي has no or one queen

→ we need to be sure that there are N number of queens on NxN board as only Last 4 constraints we can get board with no queens. So, we add new constraint to get N number of queens for Σ of X_{ij} (Note we Implicit this condition)

Implicit: $\sum_{ij} X_{ij} = N$

* ——— *

Formulation 2

- Variables : Q_K

  where K : row number ⇒ number of Q = number of rows.

- Domains : $\{1, 2, 3, \ldots N\}$

  where N : number of squares in N*N board

  4x4 board ⇒ domain from 1 to 16

- Constraints :

  Implicit: $\forall_{i,j}$ non-threatening (Q_i, Q_j)

  OR

  Explicit: $(Q_1, Q_2) \in \{(1,3),(1,4),\ldots\}$

  . . . and so on all $Q_3, Q_4 \ldots Q_{16}$

→ Note: we can consider Variables and Domain as we like but Constraints must be the same in different Formulations.

(we can't) say that one formulation is better than another but we can compare between them

Example: SudoKu

- Variables : each (open) square
- Domains : {1, 2, ---, 8, 9}
- Constraints:
  - 9-way all diff. for each column.
  - 9-way all diff for each row.
  - 9-way all diff for each region.

  (OR can have a bunch of pairwise inequality constraints)

* Constraint Graphs:

→ we can use graphs to represent this search problems.

- Unary CSP: each constraint relates one variable
- Binary CSP: each constraint relates (at most) two variables.
- Binary Constraint graph : nodes are variables, arcs show constraints.
- General-Purpose CSP algorithms use the graph structure to speed up search.

* Note : Using graphs don't make the code of constraints.

Example: The Waltz Algorithm.

- The Waltz algorithm is for interpreting line drawings of solid polyhedra (متعددة الوجوهات) as 3D objects.

- An early example of an AI computation posed (considered) as a CSP

→ we can use CSPs to draw 3D animation by determining all corners and classify each one if it is internal or external one then draw Lines.

- Approach (النهج)
  - Each intersection is a variable.
  - Adjacent intersections impose constraints on each other
  - Solutions are physically realizable 3D interpretations.

* Varieties (Classification) of CSPs

→ Varieties of Variables

[1] Discrete Variables (take discrete values from finite or infinite domains)

- Finite domains
  - Size d means $O(d^m)$ complete assignments.
  - ex: Boolean CSPs.
- Infinite domains (integers, strings, etc)
  - ex: job scheduling, variables are start/end times for each job.

[2] Continuous Variables

- ex: Start/end times for Hubble Telescope observations. (مرصد)

Linear constraints solvable in ploynomial time.

→ Varieties of Constraints ★

[1] Unary Constraints involve a single variable
(equivalent to reducing domains)
ex: SA ≠ green

[2] Binary Constraints involve pair of variables
ex: SA ≠ WA

[3] Higher-order Constraints involve 3 or more variables.

★ Preferences (Soft Constraints) الأولوية - التفضيلات
→ There is a periorty for values according to successor function
(ex: cost function)
- Often representable by a cost for each variable assignment
- ex: red is better than green in Map Coloring problem.
- Gives Constrained Optimization problems.

★ CSPs Goals:
- Find Any solution.
- Find All solutions, Find best one.

→ Standard Search Formulation of CSPs
Using the concept of Standard Search to find the best solution of CSPs problems, we use Standard Search algorithms like DFS & BFS ...

→ States defined by the values assigned so far (partial assignments)
- Initial State: the empty assignment, {}
- Successor Function: assign a value to an unassigned variable.
- Goal Test: the current assignment is complete and satisfies all constraints.

★ What would BFS do ?

→ BFS assigns level by level

. Initial State all variables are unassigned (First level)
ex: map color: all countries are not colored

. 2nd level one variable is assigned (one country has a color)

. 3rd level 2nd variable is assigned according to constraints
(1st country is red ⇒ 2nd one is green or blue)

and so on until we reach to the last level with all assigned variables.

. Successor Function: assign a value to an unassigned varriable
(ex: Color a country with red, green or blue were it!
unassigned variable : Country & Value : {red, green, blue}

. Goal Test : The current assignment is complete (all countries have colors)
and satisfies all constraints (WA ≠ NT)

→ According to this steps, we can Find the solution only at the last level so, BFS Last its advantage.
We can't use it to solve CSPs.

★ What would DFS do ?

→ DFS assigns depth child first from left to right

. Initial State all variables are unassigned
Start to assign start state then its left child,
assign the left childrens going to the depth child
(assign first node of every level then 2nd one of every level and so on according to successor Function)

. We reach to complete solution when we reach to the depth node of the first path so DFS is fast to solve CSPs
but, we need to make sure that all constraints satisfied

→ so, we add 2 ideas to DFS to be used to solve CSPs and called it Backtracking Search.

* Backtracking Search

- Depth-First Search with the next two improvements is called Backtracking Search (not the best name)

- Backtracking Search is the basic uninformed algorithm for solving CSPs.

• Idea 1: One variable at a time.
  - Variable assignments are commulative, so fix ordering.
  - We can start from any node.
    ex: WA = red then NT = green is the same as
    NT = green then WA = red
  - Only need to consider assignments to a single variable at each step

• Idea 2: Check constraints as you go
  - consider only values which don't conflict previous assignments.
  - Might have to do some computation to check the constraints
  - "Incremental goal test".

→ Using 2 ideas, when we reach to complete solution we reach to Goal Test (best solution).

• Backtracking = DFS + variable-ordering + fail-on-violation
• What are the choice points?
  1. Choose the start variable 2. Choose value to start variable
• general purpose Languages can be used to code Backtracking.

* Improving Backtracking

- General-purpose ideas give huge gains in speed.
  → make Backtracking Algorithm very Fast

we can do this by 2 ways

[1] Ordering
→ periorty for Variables
which variable should be assigned next?
→ periorty for Values of variables.
In which order should its values be tried?

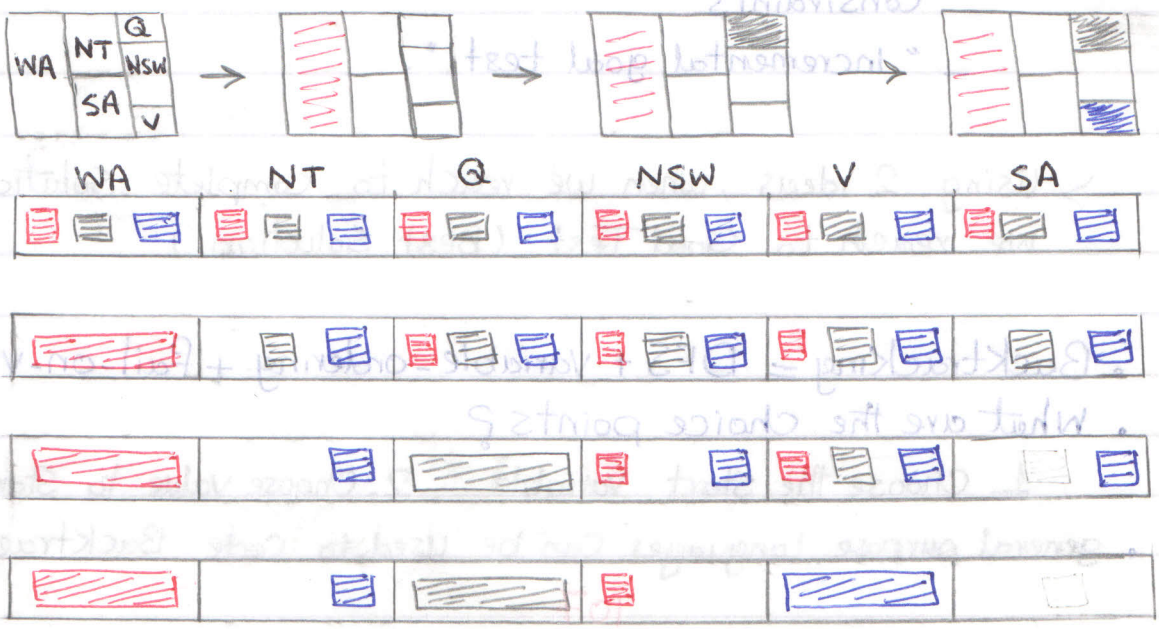
[2] Filtering:
→ detect inevitable failure early.

* ——— *

→ Filtering: Keep track of domains for unassigned variables and cross off bad options. (using Forward checking)

Forward checking: Cross off values that violate a constraint when added to the existing assignment.

Example



① ⇝ At First, Any Country can take one of 3 colors {red, green, blue}

② ⇝ IF we give WA red color, then we make forward checking to cross off red from countries next to WA ⇒ {NT, SA}

③ ⇝ Country Q can take one of 3 colors.
IF Q = green, then we make forward checking to cross off green from NT, NSW, SA (from lec draw)

④ ⇝ Country V can take one of 3 colors
IF V = blue, then we mark forward checking to cross off blue from NSW, SA
Here we can't assigne SA as All 3 Colors crossed off.

- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures

→ we mark back track to ③

- NT and SA cannot both be blue!

→ Constraint propagation method reason from constraint to constraint.
we use constraint propagation method to solve this problem (③)
→ Consistence Arc is the propagation method that we will use
→ This method know us that there is a problem early and reduce backtracking.

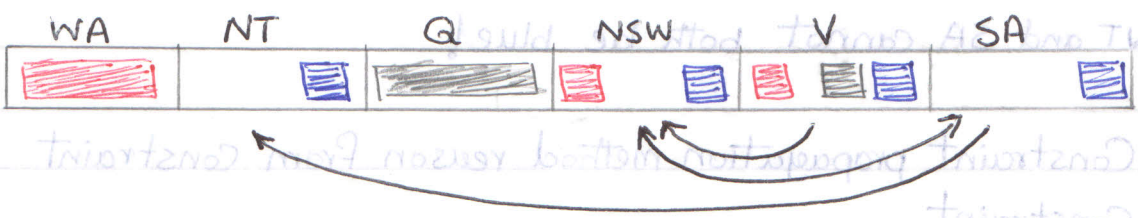
* Consistency Of A Single Arc.

An arc $X \longrightarrow Y$ is Consistent iff (if and only if) For every x (x: value of variable X) in the tail (X: tail) there is some y (y: value of variable Y) in the head (Y: head) which Could be assigned without Voilating a Constraint.

→ IF the arc is not consistent, delete the values in tail that Cause the problem and recheck the last steps.

→ Forward checking: Enforcing Consistency of arcs pointing to each new assignment.

Example: A simple form of propagation makes sure all arcs are Consistent.



1. V → NSW

   V = green ⇒ NSW = {red, blue}

   V = blue ⇒ NSW = {red}, V = red ⇒ NSW = blue

   → Arc is Consistent

2. SA → NSW

   SA = blue ⇒ NSW = red

   → Arc is Consistent.

[3] NSW ⟶ SA

NSW = red ⇒ SA = blue

NSW = blue ⇒ SA = Φ problem → delete blue from NSW

→ recheck V ⟶ NSW

V = blue ⇒ NSW = red

V = green ⇒ NSW = red

V = red ⇒ NSW = Φ problem → delete red from V

[4] SA ⟶ NT

SA = blue ⇒ NT = Φ problem Arc is not consistent

Notes

- IF X loses a value, neighbors of X need to be rechecked !
- Arc consistency detects failure earlier than forward checking.
- Can be run as a preprocessor or after each assignment.

1. assignment 2. Arc consistency 3. forward checking

→ There is no need to make more backtracting.

- Runtime : $O(n^2 d^3)$ can be reduced to $O(n^2 d^2)$

n : number of arcs 2 : number of variables

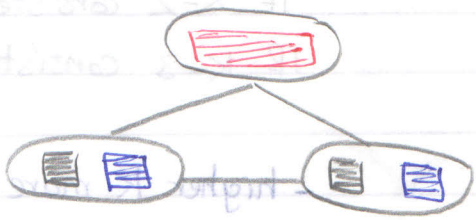
d : domain 3 : number of values.

* After enforcing arc consistency

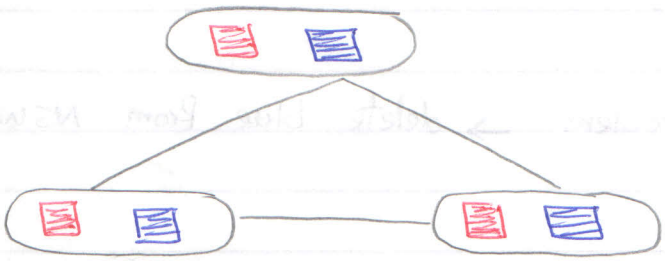
→ Can have one solution left

→ Can have multiple solutions left

→ Can have no solutions left (and not know it)



→ all arcs are consistent and there are 2 solutions.



All arcs are Consistent but There is no solution

Note: Arc Consistency still runs inside a backtracking Search.

* K-Consistency

- 1-Consistency (Node Consistency)
  each single node's domain has a value which meets that node's unary Consistraints.
  Check if a variable equal to a value or not. (Special case)

- 2-Consistency (Arc Consistency)
  For each pair of nodes, any Consistent assignment to one can be extended to the other. (we study)

- K-Consisteney
  For each K nodes, any Consistent assignment to K-1 can be extended to the K^{th} node.
  If K-2 consistent $\Rightarrow$ K-1 consistent
  If K-3 consistent $\Rightarrow$ K-2 consistent and so on.

  - higher K more expensive to compute.

Ordering

[1] Variable Ordering :

→ Choose the variable with minimum remaining values

→ Choose the variable with the fewest legal left values in its domain.

- why min rather than max ?
  → to detect failier early
  - to find errors early to reduce effort

Note: This way is the difficult but effective.

- Variable Ordering is called MRV : minimum remaining values and also "most constrained variable"

[2] Value Ordering : least constraining value

→ Given a choice of variable, choose the least constraining value.

→ The one that rules out the fewest values in the remaining variables.

→ Note that it may take some computation to determine this